NUCLEAR SCIENCE AND TECHNIQUES **25**, 010501 (2014)

Research on GPU-accelerated algorithm in 3D finite difference neutron diffusion calculation method*

XU Qi (徐琪),[1,†] YU Gang-Lin (余纲林),[1] WANG Kan (王侃),[1] and SUN Jia-Long (孙嘉龙)[1]

[1]*Department of Engineering Physics, Tsinghua University, Beijing 100084, China*

(Received March 14, 2013; accepted in revised form September 20, 2013; published online February 20, 2014)

In this paper, the adaptability of the neutron diffusion numerical algorithm on GPUs was studied, and a GPU-accelerated multi-group 3D neutron diffusion code based on finite difference method was developed. The IAEA 3D PWR benchmark problem was calculated in the numerical test. The results demonstrate both high efficiency and adequate accuracy of the GPU implementation for neutron diffusion equation.

Keywords: Neutron diffusion, Finite difference, Graphics Processing Unit (GPU), CUDA, Acceleration

DOI: 10.13538/j.1001-8042/nst.25.010501

I. INTRODUCTION

In the field of reactor physics, numerical solutions of 3-dimentional neutron diffusion equation are always required. Compared with the coarse mesh nodal techniques, the finite difference method is considered simpler and more precise, however, it costs unendurable computer time when analyzing a full-size reactor core.

Since 2006, NVIDIA's GPUs (Graphics Processing Units) has provided us with tremendous computational horsepower because of the release of CUDA [1]. In the field of nuclear reactor physics, the importance of the GPU+CPU heterogeneous platform has been growing gradually. Prayudhatama *et al.* [2] implemented a 1-D finite difference diffusion code on GPUs in 2010, and obtained up to 70× speedup compared to a corresponding CPU code. In 2011, Kodama *et al.* [3] ported the code SCOPE2 to GPUs, they got about 3 times speedup. In the same year, Gong *et al.* [4] exploited the parallelism of GPUs for the S_n code Sweep3D, which was speeded up by about 2 to 8 times.

In this work, a GPU-accelerated multi-group 3D neutron diffusion code based on finite difference method was implemented and optimized. The IAEA 3D PWR benchmark problem [5] was utilized to prove the high computational efficiency and accuracy of the GPU version code. The result in this work shows a bright future of GPU applications in nuclear reactor analysis.

II. NEUTRON DIFFUSION EQUATION

According to the neutron diffusion theory, we have the multi-group neutron diffusion equation [6] as below,

$$-\nabla\cdot D_g\nabla\varphi_g+\Sigma_t^g(r)\varphi_g=\sum_{g'=1}^{G}\Sigma_s^{g'g}\varphi_{g'}+\frac{\chi_g}{k_{\text{eff}}}\sum_{g'=1}^{G}\nu\Sigma_f^{g'}\varphi_{g'}, \tag{1}$$

where g is the energy group number, ranging from 1 to G, k_{eff} is the effective multiplication factor, and φ_g is the g^{th} neutron flux. We solve this equation by the source iteration methodology [6], which includes inner and outer iterations. The inner iteration computes a group of linear algebra equations in the form of $\mathrm{A}X=\mathrm{B}$ given the neutron scattering source and fission source. In the outer iteration, we use the neutron flux to update the neutron source and get ready for a next inner iteration.

In this work, we focus on accelerating the inner iteration. Suppose the neutron source on the right hand of Eq. (1) is known, then after discretization on XYZ grid, we can get Eq. (2), which is a linear equation with a 7 diagonal positive definite matrix as the coefficient.

$$\begin{aligned}a_{i,j,k}\varphi_{i-1,j,k}+b_{i,j,k}\varphi_{i,j-1,k}+c_{i,j,k}\varphi_{i,j,k-1}+d_{i,j,k}\varphi_{i,j,k}\\+e_{i,j,k}\varphi_{i+1,j,k}+f_{i,j,k}\varphi_{i,j+1,k}+h_{i,j,k}\varphi_{i,j,k+1}\\=s_{i,j,k},1\le i\le I,1\le j\le J,1\le k\le K.\end{aligned} \tag{2}$$

Eq. (2) is a large-scale sparse matrix problem for full-size reactor analysis. In the perspective of numerical mathematics, Jacobi iteration is inefficient, thus, arithmetic techniques, such as CG, SOR, LSOR, ADI (Alternating Direction Implicit method), are needed for efficient calculation.

III. GPU IMPLEMENTATION DETAILS

In order to test the computational capacity of GPUs, we do not resort to any mathematical skill; instead, the Jacobi iteration method is adopted for the inner iteration. The inner iteration is ported to a GTX TITAN GPU. In the inner iteration, the neutron flux is estimated according to the neutron source calculated with the neutron flux of the last source iteration. The outer iteration is still remained on CPUs to calculate the neutron effective multiplication factor according to the neutron fission source from GPUs. After an outer iteration, the effective multiplication factor is transferred from the CPU memory to the GPU memory to get the neutron source for the next source iteration. Fig. 1 shows the tasks distribution and data transfer between GPUs and CPUs during one source iteration.

* Supported by the 973 Program (No.2007CB209800) and National Natural Science Foundation of China (No.11105080)

† Corresponding author, q-xu09@mails.tsinghua.edu.cn

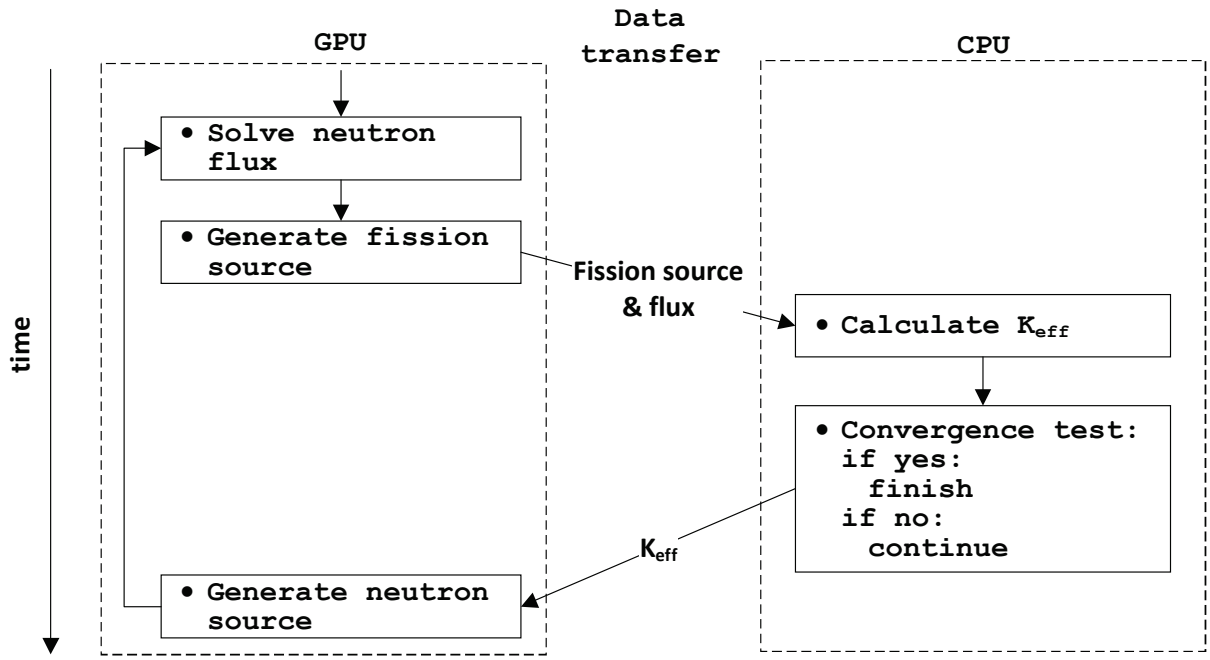

Fig. 1. Tasks distribution and data transfer between GPUs and CPUs.

A. Solving neutron flux

The neutron flux is solved via Jacobi inner iteration on GPUs. Because of the natural parallelism of the Jacobi iteration, it is of high possibility to implement this algorithm on GPUs with exciting speedups.

According to CUDA, GPUs have two levels of parallelism, the first level is called grids of thread blocks, while the second level is blocks of threads. One thread block is designed to be mapped to an SM (Streaming Multiprocessor) on GPU chips, and one thread to be mapped to an SP (Streaming Processor) in an SM. In Jacobi iteration, the main part of computing tasks are production and addition operations at each flux point, which is shown by Eq. (2). To speed up such kind of iterations, the operations at each flux point (i, j, k) should be allocated to a specific GPU thread so that the computation tasks can be spread among the SPs on GPUs. Fig. 2 demonstrates the mapping relationships between flux points and threads. As can be seen in Fig. 2, one flux point is mapped to one GPU thread, and each thread is responsible to update the flux at that point.

For a large-scale 3D reactor model, there will be millions or even tens of millions of flux points needed to be updated using the surrounding old flux, however, the hardware resources of a GPU chip is limited to create as many threads as the flux points. To solve this problem, we update the neutron flux layer by layer as is illustrated in Fig. 2. After that, there will be enough computing resources for a GPU to accelerate the inner iteration procedure for each layer of flux points.

B. Generating sources and data movements

When the neutron flux is solved after inner iterations, the fission source and the neutron source can be determined by the following equaptions:

$$S_{\text{fission}} = \sum_{g'=1}^{G} \nu\Sigma_f^{g'}\varphi_{g'}, \tag{3}$$

$$S_{\text{neutro},g} = \frac{\chi_g}{k_{\text{eff}}} S_{\text{fission}} + \sum_{g'=1}^{G} \Sigma_s^{g'g}\varphi_{g'}, \tag{4}$$

where, S_{fission} stands for the fission source and $S_{\text{neutron},g}$ stands for the neutron source of energy group g, both of which are calculated by the neutron flux newly updated. In order to reduce data exchange between the CPU and GPU memories, these two sources are obtained on GPUs in parallel.

As shown in Fig. 1, there are three data movements during one source iteration. The first data transfer happens after fission source was created, which moves fission source from device memory to host memory to calculate the effective multiplication factor k_{eff} by accumulating the fission source of each flux point. The second data movement is for comparison between old and new neutron flux, during which the new neutron flux is transferred from device to host. The third one transfers k_{eff}, a double type variable, back to device memory to get the neutron source.

C. Data storage

For the fine grid finite difference method, large number of flux points lead to large memory space needs. Suppose there

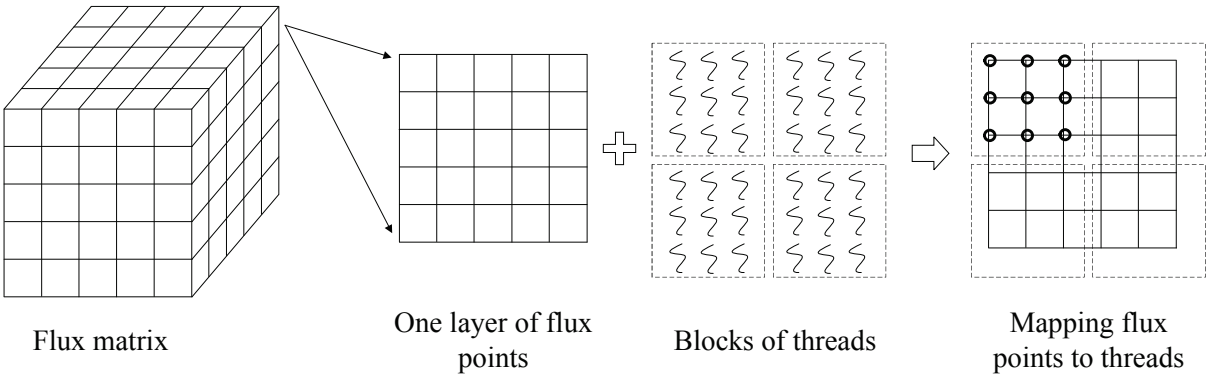

Fig. 2. Mapping relationships between flux points and threads.

are N_g energy groups, and N_x, N_y, N_z flux points in the X, Y, Z direction respectively, then the memory space to store the eight coefficients (including the neutron source $S_{i,j,k}$) would be $4(\text{bytes}) \times N_g \times N_x \times N_y \times N_z \times 8$ bytes, and the memory for the neutron flux would be $4(\text{bytes}) \times N_g \times N_x \times N_y \times N_z$ bytes. When analyzing 3D full size reactors using GPUs, all the above data should be allocated to GPU memory, which is of limited volume. For GTX TITAN, the device memory is up to 6 GB under 64 bit operating systems.

Under CUDA, a programmer is allowed to manage 7 different kinds of memory space, among which only the global memory and the texture memory are able to be utilized to store the coefficient data and the flux data. Because texture memory has a texture cache and higher bandwidth than global memory, it is advantageous to access data frequently from it. The only limitation of texture memory is that it is read-only. Thus, the coefficient data can be filled into texture memory, and the flux data be allocated to global memory.

IV. PERFORMANCE TEST

In this section, we demonstrate the accuracy and efficiency of the GPU accelerated code. Besides, we also discuss a way of performance improvement by overclocking GPU processors.

A. Experiment platform and benchmark problem

The accuracy of the GPU version diffusion code is tested by comparing the neutron flux computed by CITATION [7]. In order to prove the efficiency of the GPU code, we measure the performance of three diffusion codes listed in Table 1. 3DFD-CPU is a serial CPU version code which uses the Jacobi iteration method for inner iterations. 3DFD-GPU is obtained by accelerating the inner iteration part of 3DFD-CPU utilizing GPUs. HYPRE-8CORE [8] is a parallel diffusion code running on an 8-core CPU. The computing hardwares of these codes are also shown in Table 1.

The IAEA PWR benchmark problem, shown in Fig. 3, is used for the numerical experiment. This is an important benchmark problem widely used to test the performance of the neutron deterministic codes. The core is composed of 177 fuel assemblies, 9 of which are fully rodded and 4 of which are partially rodded. There are 64 reflector assemblies surrounding the core. The size of the assemblies is 20 cm × 20 cm × 340 cm, while the size of 1/4 core is 170 cm × 170 cm × 380 cm.

TABLE 1. Experiment platform

Code	Computing platform
3DFD-CPU	CPU:Intel Quad Q9300 @ 2.50 GHz
3DFD-GPU	GPU: NVIDIA Geforce GTX TITAN
HYPRE-8CORE	CPU:Intel Xeon E5520 @ 2.26 GHz

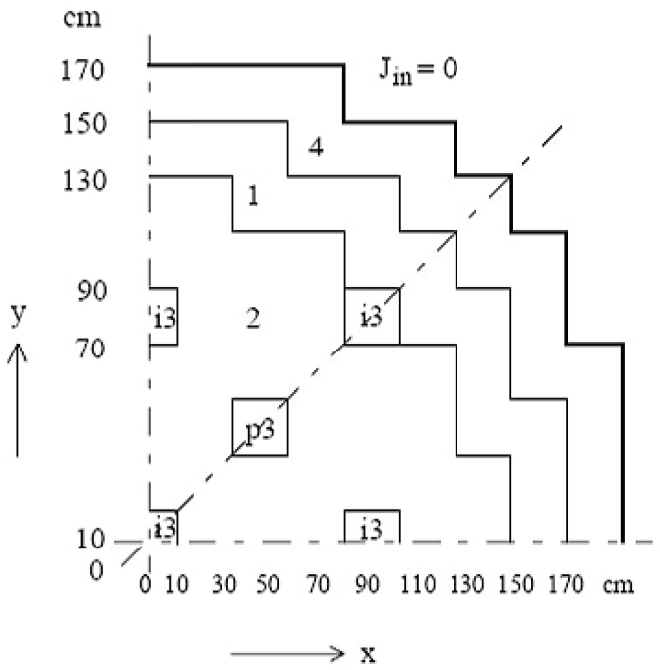

Fig. 3. Horizontal section of the core.

Benchmark Result							
3DFD-GPU(relative error %)				0.597			
CITATION(relative error %)				-2.394			
				-2.629			
			0.476	0.700	0.611		
			-1.119	-0.394	-2.330		
			-1.245	-0.603	-2.575		
		1.178	0.972	0.923	0.866		
		0.043	0.145	-0.246	-1.221		
		0.099	0.089	-0.421	-1.467		
	1.368	1.311	1.181	1.089	1.000	0.711	
	-0.167	0.001	0.038	0.001	-0.054	-0.834	
	0.149	0.138	0.019	-0.157	-0.303	-1.130	
1.397	1.432	1.291	1.072	1.055	0.976	0.757	
0.055	-0.041	0.063	0.344	0.293	0.280	0.644	
0.338	0.220	0.221	0.344	0.136	0.024	0.334	
0.729	1.281	1.422	1.193	0.610	0.953	0.959	0.777
-0.712	0.284	0.049	0.365	-0.500	0.641	0.584	0.894
-0.418	0.569	0.296	0.532	-0.491	0.476	0.323	0.579

Fig. 4. Power distribution comparison of 3DFD-GPU and CITATION (grid size =2 cm).

B. Accuracy of the GPU code

To prove the accuracy of GPU computation, we compare the power distribution of 3DFD-CPU with that of CITATION. CITATION, developed by ORNL, is an industrial class code for solving the neutron diffusion equation. The comparison results are shown in Fig. 4. The convergence criterion is set so that the simulation comes to an end when the effective multiplication factor relative error is less than 1.0×10^{-6} and the maximum point flux relative error is less than 1.0×10^{-5}. The computing grid size used in Fig. 4 is 2 cm, there are 1 372 750 spatial flux points.

In Fig. 4, the relative error stands for the difference of the code result from the benchmark result. The power distribution of the GPU version code is close to that of CITATION. The accuracy comparison tells us that there is no need to worry about the accuracy and reliability of GPUs.

C. Efficiency of the GPU code

We use the codes listed in Table 1 to testify the computing power of GPUs. Firstly, 3DFD-GPU is compared with the 8-core CPU parallelized code HYPRE-8CORE, and then a comparison of computing time between 3DFD-GPU and 3DFD-CPU is made.

According to Ref. [8], the author utilized the MPI-based parallelized linear algebra library HYPRE [9] to accelerate the diffusion code. Here we call the corresponding code in Ref. [8] as HYPRE-8CORE. HYPRE is a library developed by LLNL for solving large sparse linear systems of equations on massively parallel computers. On an 8-core tower server, the inner iteration part of the diffusion code is accelerated by the parallelized Conjugate Gradient algorithm. During simulation, the computing grid size is set to be 2.5 cm, and the convergence standard is that K_{eff} relative error converges to 1.0×10^{-5} and the maximum point flux relative error to 1.0×10^{-4}. The computation speed comparison between 3DFD-GPU and HYPRE-8CORE is shown in Table 2.

TABLE 2. Efficiency comparison of 3DFD-GPU and HYPRE-8CORE

Code	Computing time (s)	Speedup
HYPRE-8CORE	23.7	3.2
3DFD-GPU	7.5	

In Table 2, although the HYPRE-8CORE is accelerated by an 8-core server, 3DFD-GPU performs better.

The performance comparison of 3DFD-GPU and 3DFD-CPU is shown in Fig. 5. We use six kinds of grid sizes, from 5 cm × 5 cm × 5 cm to 1 cm × 1 cm × 1 cm, to demonstrate the acceleration characteristic of GPUs for Jacobi iteration. Table 3 lists the grid sizes and the corresponding grid numbers. The convergence criterion is that K_{eff} relative error converges to 1.0×10^{-6} and the maximum point flux relative error to 1.0×10^{-5}.

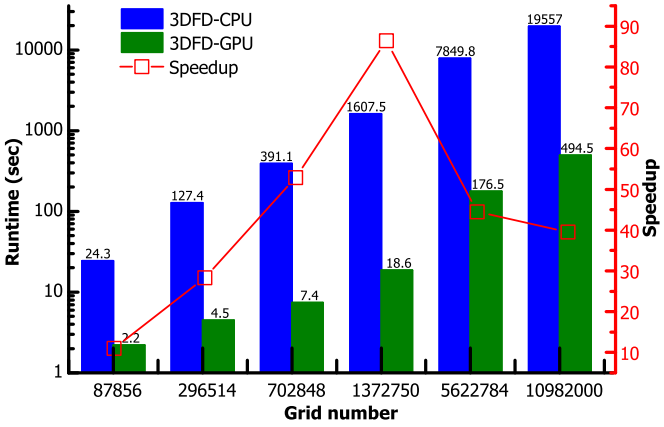

Fig. 5. (Color online) Performance comparison between 3DFD-GPU and 3DFD-CPU.

TABLE 3. Grid sizes and spatial grid numbers

Grid Size (cm)	Grid Number
5	87856
10/3	296514
2.5	702848
2	1372750
1.25	5622784
1	10982000

Figure 5 shows the amazing accelerating power of GPUs compared with CPUs, especially when the grid size is set to 2 cm, a speedup factor of 86 was obtained. This phenomenon is caused by latency hiding, when the problem scales up and the amount of data increases, all cores on GPU are working at full capacity, then data transfer from the GPU memory by part of thread blocks can be operated while other blocks are executing the computational task. However, as can be seen from Fig. 5, it should be noticed that oversized data amount may decrease the speedups, because the communication overhead between the host and the device increases and the sequential part of the code may play an increasingly important role in the whole process.

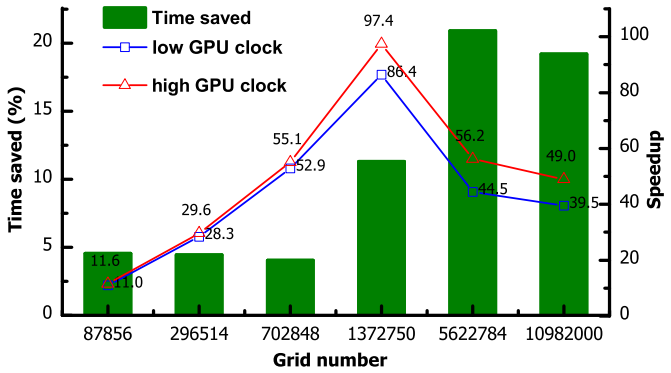

Fig. 6. (Color online) Performance improvement of overclocked GPUs.

D. Performance improvement by overclocking

In order to get the same performance with lower energy consumption, NVIDIA decreased the base clock of GPUs of Kepler series, while increased the number of streaming processors in streaming multiprocessors (SMX). The base core clock of GTX TITAN is 837 MHz, which is lower than that of GTX 580 (Fermi architecture, 1544 MHz). We use the overclocking utility NVIDIA Inspector to set the core clock to be 1166 MHz and the memory clock to be 3334 MHz. Fig. 6 shows the performance improvement after overclocking. In Fig. 6, the runtime and the speedup factor of 3DFD-GPU before and after overclocking are compared with each other, where the speedup factor is relative to the runtime of 3DFD-CPU.

Through overclocking, the GPU acceleration effect is improved. The performance improvement depends on the scale of the analyzed problem, that is to say, more obvious performance enhancement can be obtained when the grid number increases.

V. CONCLUSION

In this work, a GPU-accelerated multi-group 3D neutron diffusion code based on finite difference method was developed to speed up the finite difference methodology and examine the performance of GPUs. The IAEA 3D PWR benchmark problem is used as the problem model in the numerical experiment. By comparing the power distribution obtained from 3DFD-GPU and CITATION, we prove the accuracy of GPU computing. The performance advantage of GPUs is also demonstrated by comparing the runtime of 3DFD-GPU, 3DFD-CPU and HYPRE-8CORE.

As to the future work, mathematical accelerating techniques, such as the Conjugate Gradient method and the Chebyshev extrapolation method, will be adopted to reduce the runtime of the GPU-based finite difference method to the same order of magnitude as the coarse mesh nodal methodology.

[1] NVIDIA Corporation. CUDA C Programming Guide. 2012, 3–4 and 71–75.
[2] Prayudhatama D, Waris A, Kurniasih N, *et al*. Proceedings of AIP Conference Proceedings, 2010, **1244**: 121–126.
[3] Kodama Y, Tatsumi M, Ohoka Y. Study on GPU Computing for SCOPE2 with CUDA. Proceedings of International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering (M&C2011), Brazil, 2001.
[4] Gong C, Liu J, Chi L, *et al*. GPU Accelerated Simulations of 3D Deterministic Particle Transport Using Discrete Ordinates Method. Journal of Computational Physics, 2011, **230(15)**: 6010–6022.
[5] Argonne National Laboratory. Benchmark Problem Book. ANL-7416, Suppl.2, 1977, 277–280.
[6] Duderstadt J J and Hamilton L J, Nuclear Reactor Analysis. New York (USA): John Wiley & Sons, Inc.,1976, 285–314.
[7] Fowler T B, Vondy D R, Cunningham G W. Nuclear Reactor Core Analysis Code CITATION, ORNL-TM-2496, Supplement 2. ORNL, 1972, 104–140.
[8] Wu W B, Li Q, Wang K. Parallel Solution of 3D Neutron Diffusion Equation Based on HYPRE. Science and Technology on Reactor System Design Technology Laboratory Annual Report. Chengdu, China, 2010, 35–40 (in Chinese).
[9] Lawrence Livermore National Laboratory. HYPRE User's Manual (Version 2.7.0b). 2011, 1–6.